IRIA

CENTRE DE RENNES

IRISA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tel. 954 90 20

Rapports de Recherche

N° 154

TRIS
POUR MACHINES SYNCHRONES

Robert RANNOU

Août 1982

TRIS POUR MACHINES SYNCHRONES

OU BAUDET STEVENSON REVISITED

ROBERT RANNOU *

Résumé : L'efficacité d'un algorithme parallèle dépend souvent fortement de l'architecture. Ce papier s'intéresse au choix et à l'adaptation d'algorithmes effectuant un tri de gros tableaux. Il approfondit en fait les travaux effectués par Baudet et Stevenson [Bau 78] Une classification des calculateurs en fonction des caractéristiques des processeurs et de la classe de permutations réalisables par le réseau d'interconnexion est utilisée. Finalement, les résultats d'une mise en oeuvre effective sur Propal de l'algorithme choisi pour la classe de calculateurs à laquelle Propal appartient sont donnés.

Abstract : Parallel algorithms are very dependent on the architecture, when efficiency is paramount. In this paper, we deal with the choice of algorithms which sort large arrays. This is in fact an endension of a work done by Baudet and Stevenson [Bau 78]. A Classification of machine according to the caracteristics of elementary processors and to the class of permutations which can be performed by the interconnexion network. Finally, we give the results of an implementation on Propal of the algorithm chosen for the class of machine Propal belongs to.

* IRISA Campus de Beaulieu, Av. Général Leclerc - 35042 RENNES CEDEX

Secrétariat de Rédaction : Melle F. MOINET - U.E.R. Maths et Informatique
Campus de Beaulieu
Av. Gal. Leclerc
35042 RENNES CEDEX

TABLE DES MATIERES

1. Introduction

Un grand nombre de travaux ont porté à ce jour sur l'étude des algorithmes séquentiels de tri. Les meilleurs algorithmes connus atteignent la limite théorique qui indique que le nombre de comparaisons nécessaires pour trier N nombres est de l'ordre de $N \log_2 N$, si l'on suppose les N! permutations possibles des N nombres équiprobables. Les algorithmes Quicksort [Hoa62] et Heapsort [Flo64] ont effectivement une complexité de l'ordre de $N \log_2 N$, le premier en moyenne (mais avec une complexité dans le plus mauvais des cas de l'ordre de N^2), et le second dans le plus mauvais des cas. Le choix d'un algorithme séquentiel de tri dépend de divers facteurs, entre autres, le nombre d'éléments à trier et le rapport de ce nombre avec la taille de la mémoire disponible [Knu73] [Mar71].

Un grand nombre de travaux ont également porté sur l'étude des algorithmes de tri en parallèle. La comparaison de ces travaux est rendue difficile par le fait qu'ils reposent sur de modèles d'architectures qui diffèrent les uns des autres par maints aspects. Dans ce papier, nous commençons par effectuer un survol de ces travaux en nous restreignant au parallélisme de type synchrone, que l'on peut trouver dans les machines SIMD [Fly72]. Nous mettons l'accent sur les algorithmes qui permettent de trier des suites, dont le nombre d'éléments est largement supérieur au nombre de processeurs. Nous supposerons cependant que les suites tiennent chacune dans la mémoire directement accessible par les processeurs.

La majeure partie du papier considère une classe d'algorithmes, que nous regroupons sous le nom de Baudet-Stevenson [Bau78]. Ces algorithmes, bâtis à partir d'une même idée, diffèrent de par leur adaptation aux caractéristiques des diverses machines SIMD. Baudet et Stevenson, dans leur papier, ne considèrent qu'une seule caractéristique, la classe des permutations réalisables en un seul passage par le réseau d'interconnexion. Nous distinguerons en plus trois types de calculateurs SIMD, en prenant en compte certaines capacités des processeurs élémentaires (décodage d'instructions, adressage,...).

Finalement, nous donnerons des chiffres sur l'efficacité de l'algorithme de la famille, choisi pour le calculateur Propal II.

2. Tris pour machines SIMD : le point sur les travaux

Ce ne sont pas les meilleurs algorithmes séquentiels qui offrent le plus de parallèlisme, mais des algorithmes moins bons qui ont la propriété d'être *aveugles*. Ces derniers sont des algorithmes de tri opérant par comparaison-échange d'éléments. Ils sont dits aveugles car la suite des couples d'éléments comparés est fixe pour une taille donnée de problème. Le résultat d'une comparaison-échange n'influe pas sur le devenir de l'algorithme. Il n'existe pas d'algorithmes aveugles, connus à ce jour de complexité inférieure à $N (\log_2 N)^2$.

2.1. Les meilleurs algorithmes séquentiels

Les meilleurs algorithmes séquentiels, soit se parallélisent difficilement, soit présentent des problèmes les rendant inefficaces. Prenons le cas de Quicksort (tri d'une suite de N éléments).

```
type TAB_ELEMENT is array [1..N] of ELEMENT ;

procedure TRI_RAPIDE
            (A : in out TAB_ELEMENT ; PREMIER, DERNIER : 1..N) ;
 J : 1..N ;
begin
 J := PARTITION (A, PREMIER, DERNIER) ;
 if J - PREMIER > 1 then TRI (A, PREMIER, J-1) ;  fi ;
 if DERNIER - J > 1 then TRI (A, J+1, DERNIER) ;  fi ;
end TRI_RAPIDE ;
```

La procédure PARTITION réorganise le tableau A [PREMIER..DERNIER] de telle façon que l'élément A [PREMIER] soit en position J en fin de procédure. Tous les éléments d'indice inférieur à J sont alors plus petits que A [J], et tous les éléments d'indice supérieur à J sont plus grands ou égaux à A [J].

$$\forall I,\ PREMIER \leqslant I < J,\ A[I] < A[J]$$
$$\wedge\ \forall I,\ J < I \leqslant DERNIER,\ A[J] \leqslant A[I]$$

Il est possible d'associer un processeur à chaque partition (appels récursifs). Cet algorithme semble bien se prêter à un parallèlisme de type MIMD. Il a alors une complexité de l'ordre de N. Le gain théorique est de l'ordre de $\log_2 N$. En fait, divers problèmes se posent. Lorsque le tableau A réside dans une mémoire commune, l'algorithme est ralenti par les conflits mémoires. Ceux-ci peuvent être évités, si chaque processeur dispose d'une mémoire locale. On peut alors transférer à chaque processeur son espace de travail, c'est à dire la partition sur laquelle il doit travailler. Le coût du transfert peut ne pas être négligeable.

l'algorithme Quicksort peut être parallélisé différemment. Stone [Sto78] a effectué une mise en oeuvre de Quicksort sur la machine CDC STAR 100. Cet algorithme, compte-tenu des caractéristiques de la machine, s'est avéré être à ce jour le plus rapide.

Le parallélisme porte exclusivement sur la procédure PARTITION.

- Une comparaison vectorielle est effectuée entre A [PREMIER] et chaque élément du vecteur A [PREMIER..DERNIER]. A la suite de celle-ci, on obtient deux sous-ensembles de [PREMIER..DERNIER], PLUS_PETIT qui donne tous les éléments qui sont plus petits que A [PREMIER], et PLUS_GRAND qui donne tous les éléments plus grands ou égaux à A [PREMIER].

```
PLUS_PETIT := subset I in PREMIER..DERNIER
                                    such that A [PREMIER] > A [I] ;
PLUS_GRAND := [PREMIER..DERNIER] - PLUS_PETIT ;
L := CARDINAL (PLUS_PETIT) ;
```

- Une comparaison est ensuite effectuée sur les éléments plus grands. Le résultat de la compression est rangée dans un tableau intermédiaire TEMP.

```
TEMP [L + 1..N] := COMPRESS (A [PLUS_GRAND]) ;
```

- Les éléments plus petits sont compressés en place dans le début du tableau A.

```
A [1..L] := COMPRESS (A [PLUS_PETIT]) ;
```

- Le tableau temporaire est copié dans la deuxième partie du tableau A.

```
A [L + 1..N] := TEMP [L + 1..N] ;
```

Cet algorithme nécessite de disposer d'une machine qui traite efficacement la comparaison d'un vecteur par un scalaire, la compression, le comptage de bits pour obtenir la cardinalité de l'ensemble PLUS_PETIT et la copie des vecteurs. Le coût de l'algorithme est :

$$5.875\ (N+1)\ \log_2 (N+1) + 101.875\ N + 1) - 294.2$$

à comparer avec le coût de l'algorithme séquentiel sur STAR :

$$153\ (N+1)\ \log_2 (N+1) + 132\ (N+1) - 532.9$$

(Les deux algorithmes délaissent le tri par partition, lorsque la taille des vecteurs devient inférieure à 8 ou 9). L'algorithme vectoriel a le désavantage de nécessiter 2 N espaces mémoires (tableau temporaire).

Il serait intéressant d'effectuer l'étude de cet algorithme pour d'autres machines. D'autres problèmes viennent alors se greffer, en particulier la nécessité de tenir compte du nombre de processeurs (machines tableaux) ou de la taille maximale des vecteurs manipulés (CRAY-1).

2.2. Les algorithmes aveugles

Batcher [Bat68] a présenté deux algorithmes de complexité N ($\log_2$ N) qui ont la propriété de se prêter bien à une mise en oeuvre parallèle. Il est posible à partir de chacun de ces algorithmes de construire un réseau permettant de trier N nombres en un temps de l'ordre de $(\log_2 N)^2$. Ces algorithmes seront présentés ultérieurement (en 3.3.2 pour l'algorithme de tri bitonique, et en 3.4 pour l'algorithme pair-impair). L'algorithme de tri bitonique a été modifié par Stone [Sto71] afin d'en accroître la régularité et afin d'utiliser comme seule permutation le battage parfait (*perfect shuffle*) ou une puissance du battage parfait.

Des études ultérieures ont porté sur la mise en oeuvre du tri bitonique sur une machine possèdant un réseau d'interconnexion en forme de grille (*Mesh Connected Network*). Les processeurs forment une matrice à deux dimensions et chaque processeur est connecté à ses quatre voisins dans les deux dimensions (sauf les processeurs extérieurs).

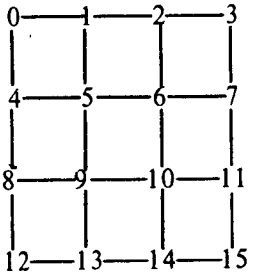


Figure 1

Orcutt [Orc74] a montré qu'il est possible de trier N^2 éléments sur une machine possédant N^2 processeurs reliés par le réseau ci-dessus en :

O (N $\log_2$ N) décalages + O $(\log_2 N)^2$ comparaisons.

Thompson et Kung [Tho76] ont amélioré les résultats d'Orcutt en modifiant la numérotation des processeurs. La numérotation est alors la suivante :

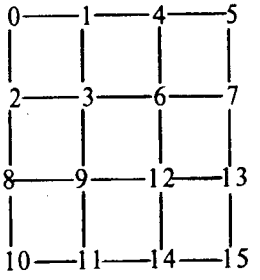


Figure 2

Cette numérotation est appelée *shuffle row-major indexing*. Elle est obtenue à partir de la numérotation précédente, appelée *row-major indexing*, en effectuant un battage parfait de la représentation binaire.

(10) 1010 devient 1100 (12)

(5) 0101 devient 0011 (3)

Le coût de l'algorithme avec cette numérotation est : $14 (N-1) - 8 \log_2 N$ décalages et $2 (\log_2 N)^2 + \log_2 N$ comparaisons. Lorsque le temps pour effectuer une comparaison est inférieur à deux fois le temps pour effectuer un décalage, cet algorithme bat l'algorithme d'Orcutt quelle que soit la valeur de N. Le gros désavantage de l'algorithme de Thompson et Kung est l'ordre très peu naturel utilisé. Sur les machines SIMD construites à ce jour, il faut compter sur un temps de sortie plus long de la suite triée.

Il est difficile de citer toutes les études portant sur les algorithmes de tri parallèle. Mentionnons cependant Muller et Preparata [Mul75] qui ont conçu un algorithme, dont l'opération élémentaire n'est pas la comparaison-échange de deux éléments. L'algorithme permet de trier N nombres en un temps de l'ordre de $\log_2 N$. Cet algorithme de par sa construction ne peut se prêter qu'à la construction d'une machine spécialisée.

Tous ces algorithmes trient N nombres avec un nombre de processeurs supérieur ou égal à N. Les algorithmes étudiés dans les chapitres suivants traitent du cas où R nombres sont triés avec N processeurs, R étant très grand par rapport à N.

3. Tris de Baudet et Stevenson

Baudet et Stevenson n'ont pas simplement donné un algorithme, mais plutôt une famille d'algorithmes bâtis sur une même idée. Cette idée, on la retrouve dans [Knu73] (Exercice 5.3.4. p. 38). La structure de ces algorithmes reflète la structure des calculateurs tableaux, qui constituent un sous-ensemble des calculateurs SIMD [Fly72]. Certains des algorithmes de la famille peuvent être cependant être adaptés aux machines pipe-line.

3.1) Les calculateurs considérés

Dans ce papier, nous entendons par calculateurs tableaux les calculateurs qui correspondent au modèle suivant : N processeurs élémentaires, capables ou non de décoder des instructions, fonctionnant en mode bloqué ou non, sont connectés à N bancs de mémoire via un réseau d'interconnexion permettant d'effectuer certaines permutations de vecteurs.

a) Les trois types de processeurs

L'étude de Baudet et Stevenson ne prend en compte que la classe de permutations réalisables par le réseau d'interconnexion. Pour notre part, nous considérons d'autres caractéristiques des calculateurs tableaux, ce qui nous permet d'en distinguer trois types. Ces caractéristiques sont la capacité ou non qu'ont les processeurs de décoder des instructions, et l'existence d'un registre d'index associé à chaque processeur.

Type I

Les processeurs de cette catégorie (NSS, Phœnix, EGPA) ont la capacité de décoder des instructions. En fait, ils possèdent pour la plupart deux modes de fonctionnement, un mode de fonctionnement SIMD pendant lequel sont effectuées les permutations et les transferts de données, et un mode de fonctionnement MIMD. Dans ce dernier mode, les processeurs exécutent généralement chacun une copie d'un même programme. A un instant donné, ils peuvent être à des étapes différentes de leur exécution. Pour sortir de ce mode les processeurs doivent se synchroniser.

Type II

Les processeurs sont incapables de décoder des instructions ; ils fonctionnent en mode bloqué, commandés par une unité de contrôle. La seule souplesse dont ils disposent provient de l'existence d'un registre d'index. Grâce à celui-ci, des éléments de vecteurs à des adresses différentes dans chaque banc de mémoire peuvent être accédés. Parmi les calculateurs de ce type, on trouve : Illiac IV, BSP.

Type III

Comme les calculateurs de type II, ils ont un mode de fonctionnement bloqué. Les processeurs élémentaires ne possèdent pas de registres d'index. Dans cette catégorie, on trouve des calculateurs conçus pour le traitement d'images : ICL DAP, MPP et Propal II. Les calculateurs pipe-line (appelés parfois vectoriels) peuvent être rangés dans cette catégorie (CRAY 1, CDC STAR 100, TI ASC).

b) Les réseaux de permutation

Comme Baudet et Stevenson, nous considérons trois types de réseaux de permutation :

- Les réseaux où les seules permutations réalisables sont des décalages dont le coût est une fonction linéaire de l'amplitude. C'est le cas du réseau de Propal II. Ces réseaux sont dits linéaires.
- Les réseaux sous forme de grille (*Mesh Connected Network*). Le réseau d'Illiac IV permet d'effectuer des décalages d'amplitude + 1, − 1, + 8 et − 8 (64 processeurs). Un décalage d'amplitude 8 a un coût inférieur à 8 décalages d'amplitude 1.
- Les réseaux capables de réaliser un battage parfait (*perfect shuffle*) en un seul passage. C'est le cas du réseau de Benes [Ben65] qu'on retrouve dans le projet Phoenix.

Quelques résultats sur le coût des permutations :

Orcutt [Orc76] a étudié le coût de mise en oeuvre des permutations à l'aide d'un réseau sous forme de grille (Illiac IV). Toute permutation d'un vecteur de N nombres peut être réalisée en $(\sqrt{N} \log_2 N)$ passages. Certaines permutations particulières, le battage parfait par exemple, peuvent être réalisées en $\sqrt{N}$ passages.

Kung et Stevenson [Kun77] ont également étudié le problème du coût des permutations, mais de façon différente. En particulier, ils ont montré qu'il est possible de transformer un algorithme qui n'utilise qu'une seule permutation α effectuée plusieurs fois, en un programme équivalent où la permutation α est remplacée par une nouvelle permutation α' nécessitant au plus quatre décalages d'amplitude 1. Il faut cependant que, préalablement à l'algorithme on effectue une permutation β sur les vecteurs manipulés et que, l'algorithme terminé, on effectue la permutation inverse β^{-1}. Les permutations β et β^{-1} reviennent à effectuer une numérotation différente des processeurs. Il est parfois possible de réduire le nombre de décalages unitaires en transformant d'une façon identique des algorithmes utilisant plusieurs permutations différentes. Il est à noter que c'est ce genre d'idée qui a conduit à l'optimisation par Thompson et Kung de l'algorithme de tri bitonique (voir 2.2).

3.2. Présentation de la famille d'algorithmes

L'idée est simple. Soit une suite de R éléments à trier. La suite est découpée en N sous-listes, chacune se trouvant dans un banc mémoire distinct. Les algorithmes de la famille comprennent deux phases.

Pendant la première phase, les processeurs trient chacun leur propre sous-liste. Il s'agit d'une phase se prêtant bien, à ce qu'on appelle la programmation verticale (MIMD) : Un même algorithme est lancé en plusieurs exemplaires, chacun ayant son propre ensemble de travail. Dans certains cas, voir 3.4, cette phase se prête également bien à la programmation horizontale ou vectorielle (SIMD).

Pendant la seconde phase, les processeurs coopèrent en vue de la fusion des sous-listes triées. Les algorithmes de fusion utilisés sont des généralisations d'algorithme de tri en parallèle de N nombres, utilisant N processeurs. La généralisation consiste à remplacer l'opération de comparaison-échange élémentaire par un algorithme de fusion de deux sous-listes triées.

Les algorithmes de la famille diffèrent de deux façons :
- par les algorithmes élémentaires de tri (phase 1) et de fusion (phase 2) utilisés (voir 3.4),
- et par l'algorithme de tri de N nombres sur N processeurs utilisé dans la phase 2 (voir 3.3).

3.3. Prise en compte des caractéristiques du réseau de permutation : choix de l'algorithme de tri de la phase 2.

Baudet et Stevenson considèrent deux algorithmes de tri, un premier algorithme, appelé tri par voisinage, qui n'utilise que des décalages unitaires, et un second algorithme, appelé tri bitonique, qui fait usage d'un battage parfait. Cette dernière permutation peut être réalisée, soit en un seul passage sur un réseau de Benes par exemple, soit en plusieurs passages sur un réseau de type ILLIAC IV. Thompson et Kung ont étudié, rappelons-le, comment améliorer l'algorithme de tri bitonique lorsqu'on dispose d'un réseau de type ILLIAC IV (voir 2.2).

3.3.1. Algorithme de tri par voisinage : décalage unitaire

a) présentation

Il s'agit d'un algorithme peu efficace de complexité N^2. Une mise en oeuvre parallèle de cet algorithme nécessite N étages de comparaisons-échanges. Supposons que les N éléments à trier sont numérotés de 0 à N – 1 et que N est pair. Il y a deux

		P0	P1	P2	P3	P4	P5	P6	P7
1	paire	27 ⟷	31	19 ⟷	81	89 ⟷	7	85 ⟷	2
	impaire	27	31 ⟷	19	81 ⟷	7	89 ⟷	2	85
2	paire	27 ⟷	19	31 ⟷	7	81 ⟷	2	89 ⟷	85
	impaire	19	27 ⟷	7	31 ⟷	2	81 ⟷	85	89
3	paire	19 ⟷	7	27 ⟷	2	31 ⟷	81	85 ⟷	89
	impaire	7	19 ⟷	2	27 ⟷	31	81 ⟷	85	89
4	paire	7 ⟷	2	19 ⟷	27	31 ⟷	81	85 ⟷	89
	impaire	2	7 ⟷	19	27 ⟷	31	81 ⟷	85	89

Tri par voisinage

Figure 3

types d'étages, les étages impairs et les étages pairs. Les premiers effectuent une comparaison-échange des couples (i, i + 1) avec i∈ [1,3,5,7,...N − 3], tandis que les seconds travaillent sur les couples (i, i + 1) pour i∈ [0,2,4,6,...N − 2]. Il y a N/2 fois, un étage pair suivi d'un étage impair.

L'algorithme de Baudet-Stevenson utilisant l'algorithme de fusion par voisinage dans la seconde phase est donné en figure 4. L'algorithme utilise deux tableaux, X et Y de taille L (le tableau X n'est rien d'autre que le tableau S à trier). Suivant que l'algorithme élémentaire de fusion de deux sous-suites opère en «place» (c'est-à-dire opère par comparaison-échange de deux éléments) ou non, l'algorithme nécessite 2 L ou 4 L emplacements mémoire.

Le choix des algorithmes élémentaires de tri et de fusion dépend du type d'architecture cible, type I, type II ou type III. Cet aspect est abordé en 3.4.

```
package TRI is
  type TABELEMENT is array [INTEGER range <>] of ELEMENT ;
  type PARTABELEMENT is array [PROCESSOR] of TABELEMENT ;
  procedure TRI_FUSION (S : in out PARTABELEMENT) is
    J : constant INTEGER := S'FIRST (2) ;
    K : constant INTEGER := S'LAST (2) ;
    L : constant INTEGER := S'LENGTH (2) ;
    MILIEU : constant INTEGER := J + L - 1 ;
    X : PARTABELEMENT renames S ;
    Y : PARTABELEMENT renames S [J..K] ;
    PAIRE : constant set of PROCESSOR := 0.. (2) PROCESSOR'LENGTH - 2 ;
    IMPAIRE : constant set of PROCESSOR := 1.. (2) PROCESSOR'LENGTH - 3 ;
    procedure TRI (A : in out TABELEMENT) is ... ;
    procedure FUSION (A ,B : in out TABELEMENT) is ... ;
  begin
    TRI (S) ;
    for M in 1..PROCESSOR'LENGTH/2 do
      Y [PAIRE] := X [ ROTATE (1) (PAIRE) ] ;
      FUSION (X [PAIRE] ,Y [PAIRE]) ;
      X [PAIRE] := Y [ ROTATE (- 1) (PAIRE) ] ;
      ----------------------------------------------------------------------------------------------------
      Y [IMPAIRE] := X [ ROTATE (1) (IMPAIRE) ] ;
      FUSION (X [IMPAIRE] ,Y [IMPAIRE] );
      X [IMPAIRE] := Y [ ROTATE (- 1) (IMPAIRE) ] ;
    end do ;
  end TRI_FUSION ;
end TRI ;
```

Figure 4

Une présentation succincte du langage utilisé est donnée en annexe.

b) Optimisation : faire travailler tous les processeurs

On peut remarquer que pendant la seconde phase, il y a au maximum N/2 processeurs actifs (cardinalité des ensembles PAIRE et IMPAIRE). Il est possible de remédier à cette inefficacité en faisant opérer tous les processeurs. L'idée est de transformer un tri par voisinage nécessitant 2 N processeurs afin qu'il fonctionne sur N processeurs. Pour celà, deux sous-listes consécutives sont installées sur le même banc mémoire (voir figure 5).

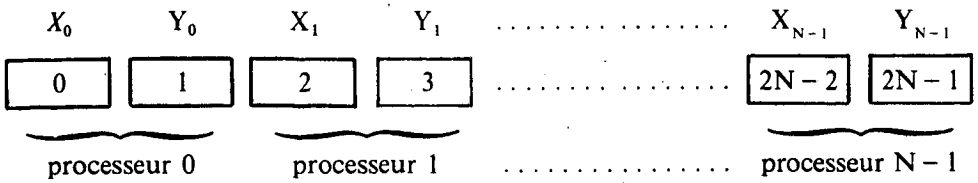
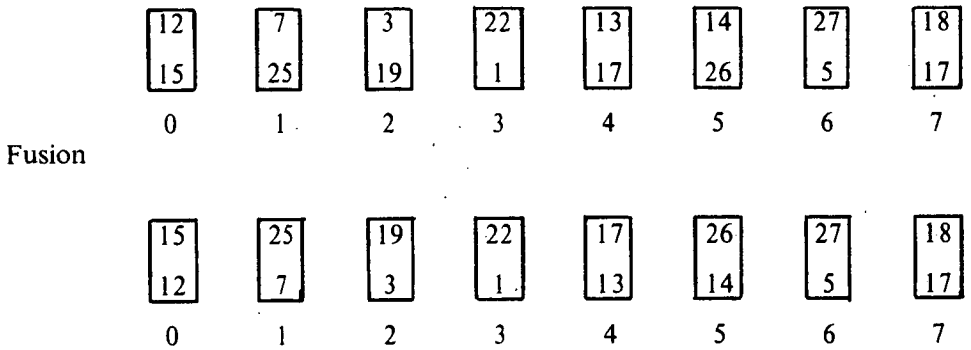
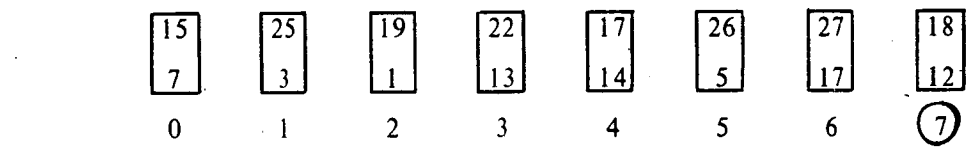


Figure 5

L'algorithme fonctionne de la façon suivante (figure 6):

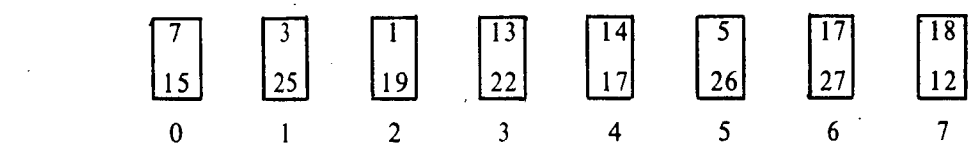


Décalage vers la gauche de la liste du bas



Fusion inverse

Le processeur 7 est inactif



Décalage vers la droite de la liste du bas

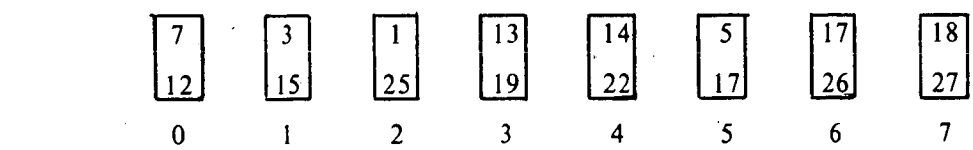


Figure 6

On suppose que la première phase de l'algorithme fournit 2 N sous-listes de taille L/2, au lieu des N sous-listes de taille L. La boucle de la deuxième phase (figure 7) est itérée N fois. A chaque itération, chaque processeur effectue une fusion des deux sous-listes se trouvant dans son banc-mémoire (ce qui correspond à la fusion des processeurs pairs du précédent algorithme), puis la première sous-liste est décalée d'une position afin que chaque processeur, sauf le dernier, effectue une fusion inverse des deux sous-listes se trouvant dans son banc (on appelle fusion inverse, un algorithme qui, étant donné deux sous-listes triées, rend deux sous-listes triées, tel que tous les éléments de la seconde sont plus petits que ceux de la première), finalement un décalage inverse est effectué toujours sur la première sous-liste (décalage, fusion inverse et décalage inverse correspondent à la fusion des processeurs impairs du précédent algorithme).

L'algorithme de Baudet - Stevenson optimisé est donné en figure 7.

```
procedure TRI_FUSION (S : in out PARTABELEMENT) is
   J : constant INTEGER := S'FIRST (2) ;
   K : constant INTEGER := S'LAST (2) ;
   L : constant INTEGER := S'LENGTH (2) ;
   MILIEU : constant INTEGER := J + L - 1 ;
   X : PARTABELEMENT renames S [J..MILIEU] ;
   Y : PARTABELEMENT renames S [MILIEU + 1..K] ;
   TOUS : constant set of PROCESSOR := full ;
   TOUS_SAUF_UN : constant set of PROCESSOR :=
                                          TOUS - PROCESSOR'LAST ;
   procedure TRI (A : in out TABELEMENT) is ... ;
   procedure FUSION (A ,B : in out TABELEMENT) is ... ;
   procedure INV_FUSION (A ,B : in out TABELEMENT) is ... ;
begin
   TRI (X) ;
   TRI (Y) ;
   for M in PROCESSOR do
      FUSION (X [TOUS] ,Y [TOUS] ) ;                    -- X<=Y

      ------------------------------------------------------------------------------------------
      X [TOUS] := X [ROTATE (1) (TOUS) ] ;
      INV_FUSION (X [TOUSSAUFUN], Y [TOUSSAUFUN] ) ;    -- X>=Y

      X [TOUS] := X [ROTATE ( - 1) (TOUS) ] ;
   end do ;
end TRI_FUSION ;
```

Figure 7

c) comparaison de l'algorithme original et de l'algorithme modifié

L'algorithme modifié nécessite N ou 2 N emplacements (tableaux X et Y) au lieu des 2 N ou 4 N emplacements de l'algorithme précédent. Voici un récapitulatif de la complexité des deux algorithmes.

Algorithme de départ	algorithme modifié
1 tri de L éléments	2 tris de L/2 éléments
N fusions de L éléments avec L éléments	2 N fusions de L/2 éléments avec L/2 éléments
2 N décalages de L éléments	2 N décalages de L/2 éléments

Les meilleurs algorithmes de tri ayant une complexité de l'ordre de $L \log_2 L$, effectuer deux tris de L/2 éléments est meilleur qu'effectuer un seul tri de L éléments. Le temps de fusion de deux listes de L éléments est de l'ordre de 2 L. Il est donc

équivalent d'effectuer N fusions de listes de taille L que d'effectuer 2 N fusions de listes de taille L/2 éléments. En fait, pour les machines de type III, les meilleurs algorithmes de fusion ont une complexité de l'ordre de $L \log_2 L$. Il est donc dans ce cas préférable d'effectuer 2 N fusions de L/2 listes de taille L/2 que N fusions de listes de taille L.

L'algorithme modifié est donc plus efficace, et de plus il a une occupation mémoire deux fois moindre.

3.3.2. Algorithme de tri bitonique : battage parfait

a) Présentation de l'algorithme

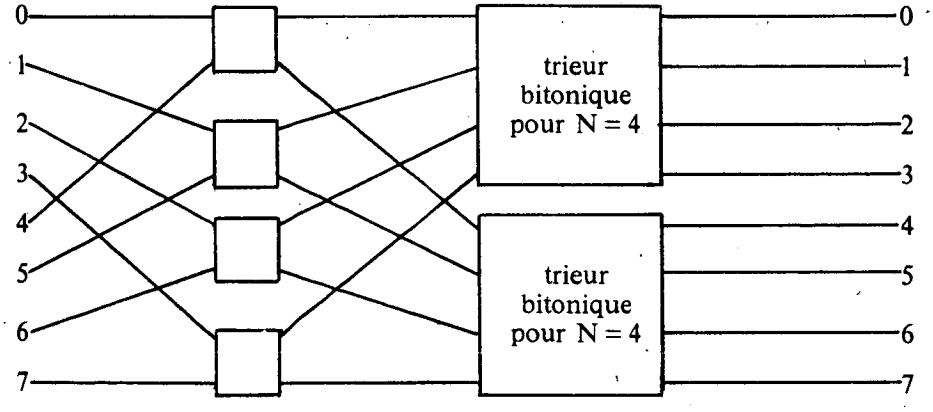
Il s'agit d'un algorithme de tri, que nous avons déjà mentionné. Il est dû à Batcher [Bat68]. Cet algorithme a ensuite été modifé par Stone [Sto72], afin d'utiliser comme seule permutation entre «étages» de comparateurs le battage parfait.

Une suite (x_i), $i = 0..p-1$ est bitonique si elle est tout d'abord strictement croissante, puis strictement décroissante. Autrement dit :

$$\exists\, k,\ 0 \leqslant k \leqslant p-1,\ (\forall i, j,\ 0 < i \leqslant j \leqslant k,\ x_i \leqslant x_j \\ \wedge\ \forall i, j,\ k < i \leqslant j \leqslant p-1,\ x_i \geqslant x_j)$$

Une suite est aussi dite bitonique, si elle peut être obtenue par rotation à partir d'une suite ayant la propriété ci-dessus (Nous supposons pour cet algorithme que la longueur des suites est une puissance de 2).

On obtient une suite triée (voir figure 8) à partir d'une suite bitonique en comparant et en échangeant tous les couples d'éléments distants de N/2. Un élément d'indice i, $0 \leqslant i < N/2$ est comparé avec l'élément d'indice $i + N/2$. Le plus petit des deux nombres se retrouve en position i, et le plus grand en position $i + N/2$. La suite $0..N/2$ est bitonique, ainsi que la suite $N/2..N-1$. De plus tous les éléments de la première sont inférieurs aux éléments de la seconde. On est alors en présence de deux suites bitoniques de taille N/2. Et on peut opérer récursivement.

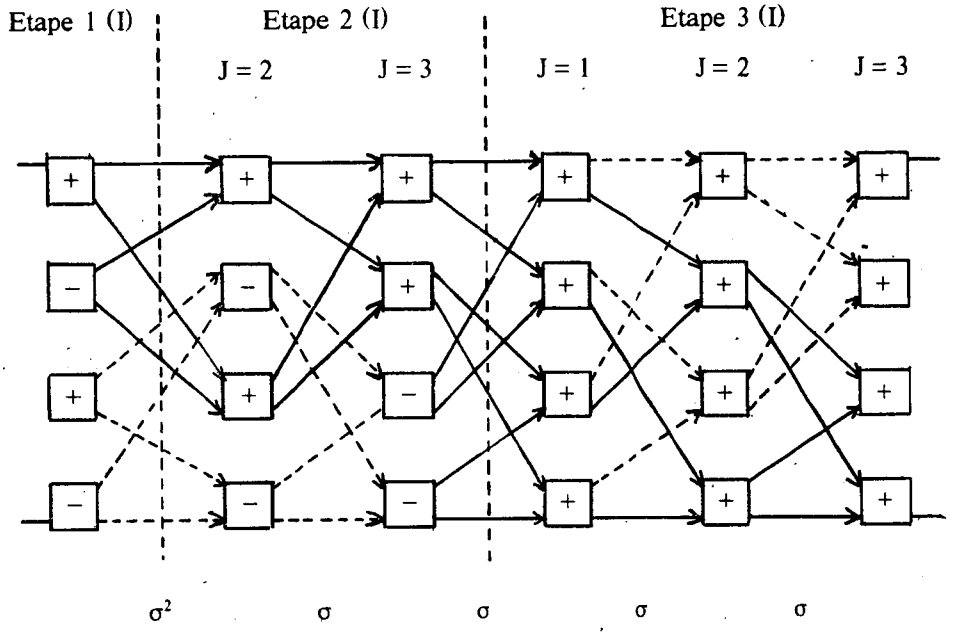


Trieur d'une suite bitonique N = 8
Figure 8

Le réseau complet opérant par tri de suites bitoniques de taille croissante est donné en figure 9.

Le premier étage du réseau de tri construit des suites bitoniques de 4 éléments. Pour celà, tous les comparateurs pairs effectuent une comparaison-échange du type plus petit en haut-plus grand en bas, tandis que les comparateurs impairs effectuent une comparaison-échange inverse. Les $\log_2 N - 2$ étapes suivantes, numérotées de 2 à $\log_2 N - 1$ construisent des suites bitoniques de taille croissante. La $I^{ème}$ étape construit $N/2^{I+1}$ suites bitoniques de taille 2^{I+1} en opérant deux par deux sur des suites bitoniques de taille 2^I. La $I^{ème}$ étape comporte I étages de comparaison-échange. La dernière étape, de numéro $\log_2 N$ trie une suite bitonique de taille N, elle comporte $\log_2 N$ étages.

La modification de Stone consiste à utiliser les puissances du battage parfait entre les étages du réseau, en effectuant une permutation des commutateurs dans chaque étage.



Réseau de tri de Batcher-Stone de 8 nombres
Figure 9

Legende de la figure 9

Les comparateurs marqués « + » envoient la plus petite de leurs entrées sur la ligne de sortie du haut et la plus grande sur celle du bas. Les comparateurs marqués « − » effectuent l'opération inverse. σ indique un battage parfait. σ^i la $i^{ème}$ puissance du battage parfait. Dans la troisième étape, les pointillés marquent l'itinéraire de la suite bitonique obtenue après les comparaisons-échanges du premier étage, en prenant tous les éléments plus petits.

Dans la troisième étape, les traits pleins marquent l'itinéraire de l'autre sous-suite formée des plus grands éléments.

L'étape 2 comporte deux «trieurs» de suites bitoniques de taille 4. Le trieur marqué en trait plein effectue un tri de telle façon que le plus petit nombre se trouve en haut. Le trieur marqué en trait pointillé effectue un tri inverse : le plus petit nombre se trouve en bas. En sortie de l'étape 2, on dispose d'une suite bitonique de taille 8.

Le programme de tri complet de Baudet-Stevenson incluant un tri bitonique dans la deuxième phase est donné en figure 10. L'algorithme élémentaire du fusion possède un paramètre supplémentaire. Lorsque ce paramètre vaut '+', la suite

résultat est croissante, sinon elle est décroissante. Le vecteur MASQUE [I] comprend 2^I '+ ', suivis de 2^I '- ', suivis de 2^I '+ ', suivis de 2^I '- ', etc... Ces masques correspondent aux commandes de la figure 9.

```
procedure TRI_FUSION (S : PARTABELEMENT) of TABELEMENT ;
  N : constant INTEGER := PROCESSOR'LENGTH ;
  LOGN : constant INTEGER := LOG (N) ;
  L : constant INTEGER := S'LENGTH (2) ; -- Longueur des sous-listes
  type COMMANDE is ( '+ ', '- ' ) ;
  type PARCOMMANDE is array [PROCESSOR] of COMMANDE ;
  type TABCOMMANDE is array [1 . . LOGN] of PARCOMMANDE ;
  MASQUE : TABCOMMANDE ;
  procedure TRI (A : in out TABELEMENT) is ... ;
  procedure FUSION (A, B : in out TABELEMENT ; SENS : COMMANDE) is ... ;
  procedure FUSION
                (S : in out PARTABELEMENT ; MASQUE : PARCOMMANDE) is
    X : PARTABELEMENT renames S ;
    Y : PARTABELEMENT [S'FIRST (2). . S'LAST (2) ] ;
    PAIRE : constant set of PROCESSOR := 0 . . (2) PROCESSOR'LENGTH - 1 ;
  begin
    Y [PAIRE] := X [ROTATE (1) (PAIRE) ] ;
    FUSION (X [PAIRE], Y [PAIRE], MASQUE [PAIRE] ) ;
    X [PAIRE] := Y [ROTATE ( - 1) (PAIRE) ] ;
  end FUSION ;
begin
  TRI (S) ;                                  -- phase 1
  FUSION (S ,MASQUE(1)) ;                    -- phase 2
  for I in 2 . . LOGN - 1 do
      for J in 1 . . LOGN do
          S [PERFECT_SHUFFLE] := S ;
      end do ;
      for J in LOGN - I + 1 . . LOGN do
          S [PERFECT_SHUFFLE] := S ;
          FUSION (S, MASQUE [J + I - LOGN]) ;
      end do ;
      for I in 1 . . . . LOGN do
          S [PERFECT_SHUFFLE] := S ;
          FUSION (S, MASQUE [LOG (L)] ;
      end do ;
  end do ;
end TRI_FUSION ;
```

Figure 10

b) Optimisation : Faire travailler tous les processeurs

Comme pour l'algorithme de tri par voisinage, il n'y a qu'une moitié des processeurs d'actifs pendant la phase 2. Modifier l'algorithme en utilisant une technique similaire à celle employée pour améliorer l'algorithme de tri par voisinage, c'est-à-dire en faisant fonctionner sur N processeurs un tri de 2 N éléments, est moins aisé.

Rappelons que les éléments sont rangés selon le schéma de la figure 5. Chaque banc mémoire contient deux sous-listes, la première est notée A , la seconde B. On dispose de deux tableaux A et B de N éléments. Le battage parfait de 2 N éléments se traduit en terme des vecteurs A et B de la façon suivante :

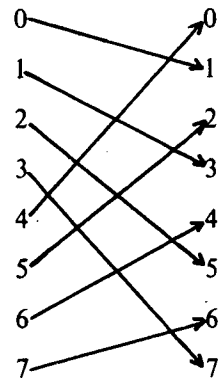
A [0] → A [0]	B [0] → A [1]
A [1] → A [2]	B [1] → A [3]
A [2] → A [4]	B [0] → A [5]
.	.
.	.
.	.
A [N/2 − 1] → A [N − 2]	B [N/2 − 1]] → A [N − 1]
A [N/2] → B [0]	B [N/2] → B [1]
A [N/2 + 1] → B [2]	B [N/2 + 1] → B [3]
A [N/2 + 2] → B [4]	B [N/2 + 2] → B [5]
.	.
.	.
.	.
A [N − 1] → B [N − 2]	B [N − 1] → B [N − 1]

Figure 11

On peut remarquer que si on constitue un vecteur S de N éléments à partir des N/2 premiers éléments de A , suivis des N/2 derniers éléments de B, et que l'on effectue un battage parfait sur le vecteur S, on obtient en position paire des éléments du vecteur A et en position impaire des éléments du vecteur B.

De même, si on constitue un vecteur Z de N éléments à partir des N/2 premiers éléments de B, suivis des N/2 derniers éléments de A, et que l'on effectue la permutation de la figure 12, notée β^n (0,1,0,1) [Len78], sur le vecteur Z, on obtient en position impaire des éléments du vecteur A et en position paire des éléments du vecteur B.

L'opération β^n (0,1,0,1) est simplement le battage parfait effectué sur un vecteur de taille 2^n (ici N) , suivi d'un échange des couples (i, i + 1) avec i pair. L'opération peut être effectuée en un passage sur le réseau de Benes en utilisant la commande récursive définie par [Len78] . Son coût sur Illiac IV est de l'ordre de $\sqrt{N}$ opérations de décalages.



β^n (0,1,0,1) = BATTAGE_PARFAIT ⊕ 1

Figure 12

L'algorithme qui simule le battage parfait de 2 N éléments sur N processeurs est le suivant :

```
procedure BATTAGE_PARFAIT_DEUX_N (A ,B : in out PARTABELEMENT) ;
  S : PARTABELEMENT [A'RANGE] ;
  Z : PARTABELEMENT [B'RANGE] ;
  assert A'RANGE is B'RANGE ;
```

```
begin
    S[0..N/2 - 1] := A [0..N/2 - 1] ;
    S[N/2..N - 1] := B [N/2..N - 1] ;
    S [PERFECT_SHUFFLE] := S ;
    Z [0..N/2 - 1] := B [0..N/2-1] ;
    Z [N/2..N - 1] := A [N/2..N - 1] ;
    Z [BETA (0,1,0,1) ] := Z ;
    A [0..(2)N - 1] := S [0..(2) N - 1] ;
    B [0..(2) N - 1] := Z [0..(2) N - 1] ;
    A [1..(2) N - 1] := Z [1..(2) N - 1] ;
    B [1..(2) N - 1] := S [1..(2) N - 1] ;
end BATTAGE_PARFAIT_DEUX_N ;
```

Figure 13

Pour obtenir le tri complet de Baudet-Stevenson modifié, il suffit :

- de remplacer log (N) par log (2·N),
- de remplacer le tri de la phase 1 par deux tris, chacun portant sur une moitié du vecteur S (TRI (X) ; TRI (Y)),
- de remplacer le battage parfait de N éléments par l'algorithme précédent effectuant le battage parfait de 2 N éléments sur N processeurs,

  BATTAGE_PARFAIT_DEUX_N (X, Y) ;
- de modifier le tableau MASQUE, de façon à ce que MASQUE [I] contienne 2^{I-1} '+', suivis de 2^{I-1} '−', suivis de 2^{I-1} '+', etc.

  ex) N = 3
  MASQUE [1] = ('+' , '−' , '+' , '−' , '+' , '−' , '+' , '−')
  MASQUE [2] = ('+' , '+' , '−' , '−' , '+' , '+' , '−' , '−')
  MASQUE [3] = ('+' , '+' , '+' , '+' , '−' , '−' , '−' , '−')
  MASQUE [4] = ('+' , '+' , '+' , '+' , '+' , '+' , '+' , '+')
- de modifier la procédure fusion : Il n'y a plus de rotations à effectuer pour amener les sous-listes impaires dans les bancs-memoires, l'algorithme peut être appelé directement, de la façon suivante :

  FUSION (X , Y , MASQUE [...]) ;

c) Comparaison des deux algorithmes

L'avantage premier de la modification est de réduire l'occupation mémoire de l'algorithme. Celle-ci passe de 2 L ou 4 L, suivant l'algorithme élementaire de fusion choisi, à L ou 2 L. Le gain en temps d'éxécution est moins évident :

Tri avec battage parfait non modifié	Tri avec battage parfait modifié
1 tri de L éléments	2 tris de L/2 éléments
$(\log_2 N \cdot \log_2 (N+1))/2$ fusions de L éléments par L éléments	$(\log_2 (N+1) \cdot \log_2 (N+2))/2$ fusions de L/2 éléments par L/2 éléments
$\log_2 N \cdot \log_2 (N+1)$ décalages unitaires de L éléments	pas de décalages
$\log_2 N \cdot \log_2 (N-1)$ battages parfaits sur L éléments	$\log_2 N \cdot \log_2 (N+1)$ battages parfaits sur L/2 éléments
	$\log_2 N \cdot \log_2 (N+1)$ β (0,1,0,1) sur L/2 éléments
	$8 \cdot (\log_2 N) \cdot \log_2 (N+1))$ transferts de L/2 éléments

(Il est possible de combiner certains battages parfaits. Les puissances du battage parfait peuvent être réalisées sur certains réseaux (Benes) en un passage. La même transformation que sur le battage parfait peut être appliquée sur les puissances.)

On peut rappeler (voir 2.2) qu'Orcutt [Orc74] a étudié la mise en œuvre de l'algorithme de tri bitonique sur une machine, où les processeurs forment une matrice à deux dimensions. Son algorithme peut être utilisé dans la phase 2. Si on accepte que le tableau trié est rangé selon le shéma *shuffle row-major* (voir 2.2), il est possible d'utiliser l'algorithme de Thompson et de Kung [Tho76] plus performant.

3.4. Prise en compte du type des processeurs : Choix des algorithmes élémentaires de tri et de fusion

Dans ce chapitre, nous distinguerons, comme nous l'avons annoncé trois types d'architectures, auxquels correspondent des algorithmes optimum différents de tri et de fusion. Les algorithmes sont des algorithmes séquentiels qui sont conceptuellement créés en plusieurs exemplaires lors de leur appel. Ils ne sont effectivement créés en plusieurs exemplaires que dans le cas des machines de type I. Pour les machines de type II ou III, une transformation en algorithmes vectoriels doit être effectuée. Baudet et Stevenson ne considèrent que les machines de type II.

3.4.1. Machines de type I

Chaque processeur étant capable de décoder des instructions, le problème consiste à choisir un algorithme séquentiel en fonction de la longueur de la liste et de la taille de la mémoire associée à chaque processeur. On peut trouver dans [Knu73] ou dans [Mar71], une comparaison d'algorithmes séquentiels de tri et de fusion selon ces critères.

Un autre critère entre en jeu. Plusieurs exemplaires du même algorithme séquentiel sont éxécutés en parallèle. Un point de synchronisation est placé en fin de chaque exemplaire. L'instruction parallèle de tri et de fusion ne se termine que lorsque tous les exemplaires ont terminé leur éxécution. Ce n'est donc pas tant le temps moyen d'éxécution de l'algorithme qui nous intéresse que la distribution des temps, voire le temps maximum.

Prenons le cas de Quicksort et de Heapsort. Knuth ([Knu73] p.380) donne les chiffres suivants pour un programme écrit pour sa machine MIX.

	Temps moyen	Temps maximum	Taille du programme	Occupation des données
Quicksort	$12.67\ N \log_2 N - 1.92\ N$	$\geqslant 2\ N^2$	63	$N + 2 \log_2 N$
Heapsort	$23.08\ N \log_2 N + 0.2\ N$	$26\ N \log_2 N$	30	N

On s'aperçoit qu'il y a peu de différence dans le cas de Heapsort entre le temps moyen et le temps maximum, alors que dans le cas de Quicksort, le temps maximum est de l'ordre de N^2 et le temps moyen de l'ordre de $N\log_2 N$. Quand bien même, Quicksort est en moyenne plus rapide qu'Heapsort, il peut être intéressant de choisir Heapsort. Une analyse en fonction de cette contrainte supplémentaire reste encore à faire.

D'autres algorithmes de tri, en particulier des algorithmes de tri par fusion (voir 3.4.2), ont également une complexité de l'ordre de $N\log_2 N$. Le temps maximum d'éxécution de l'un d'entre eux (l'algorithme *Natural two-way merge sort* [Knu73]), toujours sur la machine MIX, est de $12.5\ N\log_2 N$. L'inconvénient de ces algorithmes par rapport à Heapsort est de nécessiter deux fois plus d'espace.

3.4.2. Machines de type II et III

Les processeurs élémentaires des machines de type II ou III n'ont pas la faculté de décoder des instructions. Ils fonctionnent en mode bloqué, pilotés par une unité de commande. Pour éviter une trop grande inefficacité, dû à l'inactivité de certains processeurs, on est conduit à choisir des algorithmes élémentaires de tri et de fusion qui ne présentent pas trop de chemins d'éxécution différents.

a) Les algorithmes aveugles

Une façon extrême de régler ce problème consiste à choisir des algorithmes élémentaires ne possédant qu'un seul chemin d'éxécution possible. Les processeurs, auxquels on fournit du travail, restent actifs pendant toute l'éxécution de l'algorithme. Les algorithmes aveugles ont un avantage supplémentaire : Pour une taille donnée de problème, la suite formée des couples d'éléments à comparer et à échanger est fixe et connue à l'avance. Cette suite d'adresses, n'a pas à être recalculée, si, comme c'est le cas pour l'algorithme de Baudet-Stevenson, plusieurs activations d'un même algorithme aveugle sont lancées sur un nombre identique de données.

Nous avons vu que Batcher [Bat68] présente deux algorithmes de tri aveugle. L'algorithme de tri (voir 3.3.2) qui fonctionne par création de suites bitoniques de taille croissante, nécessite $((\log_2 L)^2 + \log_2 L)/4$ comparaisons échanges pour un tri de L éléments. L'algorithme de tri d'une suite bitonique de taille 2L nécessite $L\log_2(2L)$ comparaisons-échanges.

Le second algorithme de tri présenté par Batcher est l'algorithme pair-impair. C'est un algorithme qui opère par fusion de suites ordonnées de taile croissante. L'algorithme de fusion utilisé prend en entrée deux suites triées de taille s et t, pour donner une suite triée de taille s + t (figure 14).

Les éléments de rang pair de chacune des sous-suites sont amenés sur un fusionneur dit pair, et les éléments de rang impair sur un fusionneur dit impair. Les deux fusionneurs opèrent sur des suites triées de taille s/2 et t/2 (au problème près de la parité de s et t), pour donner en sortie une suite triée de taille (s + t)/2. Les comparateurs du dernier étage terminent le travail en comparant-échangant les éléments (i, i + 1) avec i impair et $i \leqslant s + t - 1$ (Pour une preuve de l'algorithme, (voir [Bat68]). A cette version récursive de l'algorithme, correspond une version itérative plus performante [Knu73], qui calcule la suite des indices des couples à comparer.

Le nombre de comparaisons-échanges de l'algorithme de fusion (fusion de deux suites de L éléments) est :

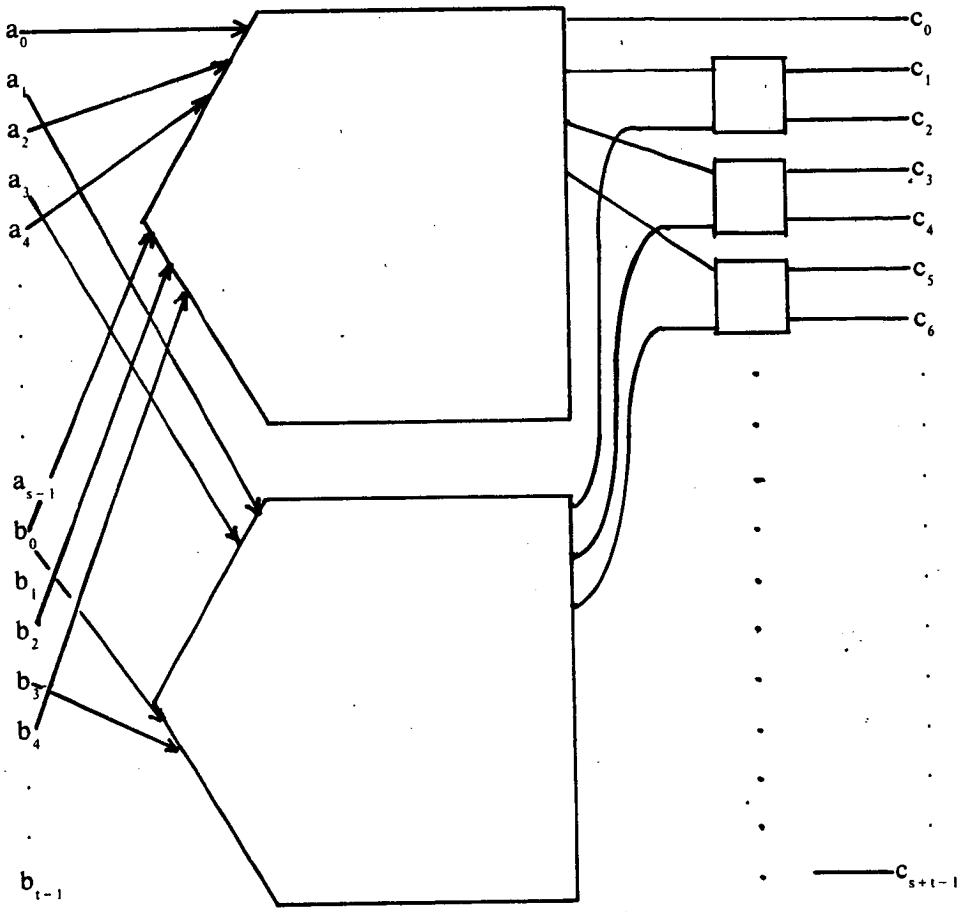
$$L\log_2 L + 1$$

Celui de l'algorithme de tri-fusion qui en découle est :

$$(((\log_2 L)^2 - \log_2 L + 4)\ L/4) - 1$$

Ce dernier est donc préférable à l'algorithme bitonique. Un algorithme de tri-fusion de 1024 éléments effectue 28160 comparaisons-échanges, tandis que l'algorithme pair-impair en effectue 24063. Ce dernier a l'avantage supplémentaire de fonctionner quelque soit le nombre d'éléments à trier, alors que le tri bitonique exige que ce nombre soit une puissance de deux. L'algorithme séquentiel de tri pair-impair est donc meilleur que l'algorithme séquentiel de tri bitonique.

Tout deux cependant, ont une complexité de l'ordre de $L(\log_2 L)^2$ à comparer à la complexité de l'ordre de $L\log_2 L$ d'algorithmes comme Quicksort ou Heapsort. Il n'existe pas d'algorithmes connus de complexité inférieure à $L(\log_2 L)^2$, ayant la propriété d'être aveugle. Le problème qui se pose est de savoir s'il n'existe pas d'algorithmes de complexité inférieure, qui bien que n'étant pas aveugles, soient plus efficaces que le tri pair-impair sur les machines de type II ou III.



Fusion pair-impair de Batcher
Figure 14

b) Algorithmes pour machines de type II

Baudet et Stevenson présentent un algorithme élémentaire qui, comme l'algorithme pair-impair de Batcher, construit des listes triées de taille croissante. Ces algorithmes ne diffèrent sensiblement l'un de l'autre que par l'algorithme de fusion utilisé.

```
procedure FUSION (A, B : TABELEMENT ;C : out TABELEMENT ) is
  W : constant INTEGER := GROSSE_VALEUR ;
  I : A'RANGE ; J : B'RANGE ;
  assert C'LENGTH = A'LENGTH + B'LENGTH - 2 ;
begin
    A [A'LAST] := W ;   B [B'LAST] := W ;
    I := A'FIRST ;   J := B'FIRST ;
    for K in C'RANGE do
      if A [I] > B [J]
        then C [K] := B [J] ;   J := J + 1 ;
        else C [K] := A [I] ;   I := I + 1 ;
      fi ;
    end do ;
end FUSION ;
```

Figure 15

Cet algorithme très simple, effectue la fusion de deux listes A et B de taille L (L = A'LENGTH - 1) et L (L = B'LENGTH - 1), et construit une liste C de taille L + L. Le nombre de comparaisons échanges est fixe et égal à L + L, du fait de

l'existence des sentinelles en dernière position des listes A et B. Une mise en œuvre vectorielle de cet algorithme sur une machine de type II a un coût de L + L comparaisons et de 2(L + L) mouvements d'éléments vers C (en supposant qu'à chaque fois, un sous-ensemble non vide des processeurs choisit la première branche de l'instruction **if**, tandis qu'un sous-ensemble également non vide choisit la seconde).

Cet algorithme a le désavantage sur l'algorithme de fusion pair-impair de Batcher de nécessiter deux fois plus d'espace. Mais, contrairement à celui donné par [Bau78], il n'exige pas que les listes A et B soient de même taille. L'algorithme de tri qui en résulte peut ainsi fonctionner sur des listes de taille quelconque, et non pas seulement comme dans [Bau78], sur des listes dont la taille est une puissance de deux.

Au lieu d'utiliser la technique de la sentinelle, l'algorithme de la figure 16 effectue un test d'épuisement de la liste d'où provient le plus petit élément d'une comparaison.

```
procedure FUSION (A ,B : TABELEMENT ; C : out TABELEMENT ) is
  I : A'RANGE := A'FIRST ;
  J : B'RANGE := B'FIRST ;
begin
  for K in C'RANGE do
    if A [I] >= B [J]
      then
        C [K] := B [J] ;
        if J = B'LAST
          then exit ;
          else J := J + 1 ;
        fi ;
      else
        C [K] := A [J] ;
        if I = A'LAST
          then exit ;
          else I := I + 1 ;
        fi ;
    fi ;
  end do ;
  if I < A'LAST
    then C [K + 1 . . C'LAST] := A [I . . A'LAST] ;
    else C [K + 1 . . C'LAST] := B [J . . B'LAST] ;
  fi ;
end FUSION ;
```

Figure 16

La vectorisation de cet algorithme illustre bien le manque d'efficacité qui peut résulter d'un trop grand nombre de chemins d'éxécution possibles. Considérons la fusion de deux listes de taille s et t. Sur une machine de type II, la boucle tend à être éxécutée s + t fois, d'où s + t comparaisons vectorielles, 2(s + t) mouvements de données et tests d'épuisement, min (N, 4(s + t)) inhibitions de processeurs (traduction de linstruction **exit**). Au pire des cas, s + t mouvements de données sont de plus nécessaires pour éxécuter l'instruction **if** qui suit la boucle. Le mode de fonctionnement bloqué pénalise cet algorithme par rapport à celui avec sentinelle.

La propriété de disposer d'un registre d'index pour chaque processeur est ici essentielle. Sans elle, il est impossible d'éxécuter en une seule instruction vectorielle les tests et les affectations, puisque les variables apparaissant en position d'indice de tableau sont propres à chaque activation de la procédure FUSION.

c) Comparaison des coûts des algorithmes de fusion

Fusion de deux listes de L éléments

Fusion avec sentinelle type II	Fusion pair-impair type III
2L boucles comprenant 1 comparaison 2 copies d'éléments 2 incrémentations Occupation mémoire : 4L	$(L\log_2 L) + 1$ boucles comprenant 1 comparaison 1 échange Occupation mémoire : 2L

L'algorithme de fusion avec sentinelles est meilleur que l'algorithme pair-impair pour L suffisamment grand.

d) Un problème posé par le tri bitonique de la phase 2

L'algorithme utilisant le tri bitonique dans la seconde phase (voir 3.3.2) présente un problème supplémentaire. L'algorithme de fusion élémentaire (figure 10) possède un paramètre indiquant si la liste résultat doit être croissante ou décroissante. Tout les processeurs n'ont pas à effectuer le même travail. Dans le cas des machines de type I, celà ne pose évidemment aucun problème. Dans le cas des machines de type III, il y a perte d'éfficacité. On doit dans un premier temps ne faire fonctionner que les processeurs qui doivent effectuer une fusion croissante, puis dans un second temps seulement ceux qui doivent effectuer une fusion décroissante. La vectorisation des algorithmes de fusion avec sentinelles pose peu de problèmes. Dans l'algorithme de la figure 15, seule la boucle **for** doit être remplacée par une boucle **while** permettant un cheminement différent dans l'ensemble d'indices C'RANGE.

Récapitulatif

Choix des algorithmes élémentaires de tri

	Type I	Type II	Type III
Fusion L.L coût	Avec sentinelle Figure 15 2L	Avec sentinelle Figure 15 2L	Batcher pair-impair Figure 14 $L\log_2 L + 1$
Tri L coût	Heapsort $L\log_2 L$	Tri-fusion sentinelle $L\log_2 L$	Tri-fusion pair-impair $((\log_2 L)^2 - \log_2 L + 4)L/4 - 1$

Figure 16

4. Tri pour Propal

Dans les chapitres précédents, nous avons montré en quoi le choix d'un algorithme basé sur l'idée de Baudet et de Stevenson, dépend du type de processeurs et de la classe des permutations réalisables efficacement par le réseau d'interconnexion. Dans le cas des machines de type III ne possédant comme réseau qu'un réseau capable d'effectuer des décalages circulaires, nous avons montré que le meilleur algorithme de la famille utilise :

-dans la seconde phase, un tri par voisinage,
-dans la seconde phase, comme algorithme de fusion de sous-listes, l'algorithme de Batcher pair-impair,
-dans la première phase, comme algorithme de tri, l'algorithme de tri-fusion basé sur l'algorithme de fusion de Batcher pair-impair.

Cet algorithme a été programmé sur Propal II, machine de type III. La configuration dont on dispose possède 64 processeurs élémentaires. Chacun d'entre eux dispose d'une mémoire de travail de 256 bits et d'une mémoire d'extension de 16K bits qui contient le tableau à trier. Dans l'étude du coût de l'algorithme, nous ferons cependant varier le nombre de processeurs et la taille de la mémoire d'extension.

Le coût estimé de l'algorithme est le suivant (L représente le nombre d'éléments dans chaque banc-mémoire. Le nombre total d'éléments à trier est R = N L. Bien que l'algorithme fonctionne quelque soit L multiple de 2, les coûts ne sont donnés que pour L puissance de 2) :

$2((\log_2 L - 1)^2 - \log_2 L + 5)L/8 - 1$
opérations de comparaisons-échanges -phase 1
$2N(L - 1)(\log_2 - 1) + 2N$
opérations de comparaisons-échanges -phase 2
2N décalages de L/2 éléments -phase 2

La première phase a donc une complexité de l'ordre de $L(\log_2 L)^2$, tandis que la seconde phase a une complexité de l'ordre de $NL\log_2 L$. La complexité de cet algorithme est à comparer à celle des meilleurs algorithmes séquentiels connus. La différence n'est pas importante. Seule une mise en œuvre effective de l'algorithme peut montrer s'il y a gain de performance et, si oui, de quel ordre il est. Il est à noter cependant que les résultats obtenus sur Propal II ne peuvent être généralisés aux autres machines de type III qu'avec précaution.

Rappelons que les algorithmes élémentaires de fusion et de tri, choisis sont des algorithmes aveugles. Pour une taille donnée de problème, la suite formée des couples d'adresses à comparer est fixe et peut n'être calculée qu'une seule fois. Les 2N fusions et les 2 tris reçoivent en paramètre cette suite d'adresses. Le calcul de ces adresses est purement séquentiel, il peut être effectué par le calculateur-hôte auquel est connecté Propal, avant que le tri véritable soit lancé sur Propal. Le coût d'obtention de la suite d'adresses n'a donc pas été pris en compte.

En remarque, on peut noter que cette caractéristique des algorithmes élémentaires est également intéressante dans le cas où on doit effectuer un tri sur un gros tableau ne tenant pas en mémoire d'extension. Une solution consiste à diviser le tableau en un nombre minimal de sous-tableaux de taille égale (quitte à compléter certains sous tableaux de valeurs très grandes ou très petites) et à effectuer un tri de Baudet-Stevenson sur chaque sous-tableau. Le soin de compléter le tri en fusionnant les sous-tableaux est alors laissé au calculateur-hôte.

Le tableau de la figure 17 compare le coût de l'algorithme Heapsort avec celui de l'algorithme de Baudet-Stevenson sur Propal avec 64, 1024 et 16384 processeurs. Le coût de l'algorithme Heapsort est un coût moyen d'exécution sur Mitra 125, calculateur-hôte de Propal, obtenu à l'aide des formules données par Knuth ([Knu72] p.p. 148-149). Les coûts de l'algorithme de Baudet-Stevenson ne tiennent pas compte des transferts du tableau entre la mémoire d'extension et celle du calculateur-hôte, avant et après le tri.

R : Nombre d'éléments à trier	Heapsort	Baudet-Stevenson		
		64	1024	16384 MPP
$256 = 2^8$	137			
$512 = 2^9$	242	382		
$1024 = 2^{10}$	540	993		
$2048 = 2^{11}$	1192	2462		
$4096 = 2^{12}$	2606	5915		
$8192 = 2^{13}$	5657	13826	2163	
$16384 = 2^{14}$	12201	31690	6010	
$32768 = 2^{15}$	26178	71530	15628	
$65536 = 2^{16}$	55904	158558	38711	
$131072 = 2^{17}$	118906	352103	92598	34585
$262144 = 2^{18}$	252007	770955	215785	96070
$524288 = 2^{19}$	532403	1871572	493020	249783
$1048576 = 2^{20}$	1121582		1109241	618694
$2097152 = 2^{21}$	2356716		2464307	1479514

Comparaison de l'algorithme Heapsort sur Mitra
et de l'algorithme de Baudet-Stevenson sur Propal

(Les coûts sont donnés en millisecondes. Les éléments à trier contiennent 16 bits de données et 16 bits de clé. Le trait indique quand il y a dépassement de capacité-mémoire de notre configuration, 16K bits.)

Figure 17

On peut remarquer qu'avec 64 processeurs, l'algorithme de Baudet-Stevenson est toujours plus mauvais qu'Heapsort et que ce dernier est d'autant meilleur que le nombre d'éléments à trier est important. Le gain en rapidité (speed-up) en faveur d'Heapsort varie de 1.28 pour R = 256 à 3.5 pour R = 2097152. Pour N = 1024, l'algorithme de Baudet-Stevenson est meilleur qu'Heapsort pour R compris entre 8192 et 1048576 (gain variant de 2.6 à 1.01). Heapsort devient meilleur pour des tailles plus grandes. Pour N = 262144, le tri de 2097152 éléments ne se fait que 4.2 fois plus vite!

Les transferts entre mémoire d'extension et mémoire du calculateur-hôte entraine un sur-coût compris entre 1.5 % lorsque peu d'éléments se trouvent dans chaque processeur et 2 ‰ pour les gros tris.

5. Conclusion

La conception et la mise en œuvre d'un algorithme pour machine SIMD dépend fortement des caratéristiques de la machine-cible, entre autres la capacité ou non qu'ont les processeurs de décoder des instructions, l'existence ou non d'un registre d'index associé à chaque processeur et la classe de permutations réalisables par le réseau d'interconnexion. Dans le cas du tri, on s'aperçoit que la seule absence d'un registre d'index peut pénaliser fortement une machine. C'est le cas de Propal.

Il serait intéressant d'étendre cette étude au tri de gros tableaux ne tenant pas dans la mémoire des processeurs. Cette étude ne peut se faire que cas par cas. L'algorithme que nous avons déjà mentionné (voir 4), où les processeurs parallèles effectuent le tri de sous-tableaux tenant en mémoire, qui sont ensuite fusionnés par le calculateur-hôte, devrait se comporter honorablement dans la majeure partie des cas, les transferts entre la mémoire des processeurs et le calculateur-hôte n'y étant pas trop important.

6. Annexe : Langage utilisé

Le langage que nous sommes actuellement en train de définir, est inspiré d'Ada [Led81]. Il s'en distingue principalement par l'utilisation des tableaux, structures priviligiées sur lesquelles s'applique le parallélisme. Dans l'exemple de la figure 4, PROCESSOR est un type indice parallèle prédéfini et INTEGER **range** <> est un type indice séquentiel à bornes non connues à la compilation. S est donc un tableau parallèle dont les éléments sont des tableaux séquentiels à bornes non connues, mais toutes égales.

Comme en Pascal, il y a le type ensemble . On peut à l'aide de ceux-ci opérer sur des sous-ensembles d'éléments de tableaux.

X [PAIRE] := Y [PAIRE] ;
est équivalent à (N est le nombre de processeurs) :
X [0] := Y [0] ; X [1] := Y [1] ; .... X [N – 2] := Y [N – 2] ;

Dans la figure 13, la notation : [1 . .(2) N – 1] indique un sous-ensemble d'éléments de PROCESSOR (indice du type tableau PARTABELEMENT), compris entre 1 et N – 1, et de la forme 1 + 2i (i entier naturel).

On peut effectuer des permutations : X [ROTATE (1) (PAIRE)]. Un décalage circulaire de 1 est effectué sur l'ensemble d'indices PAIRE, ce qui revient à effectuer un décalage inverse du tableau X. La notation X [PERFECT_SHUFFLE] indique un battage parfait sur le tableau X.

Le concept de multi-action est utilisé [Ban81]. La procédure TRI opére sur un tableau de type TABELEMENT. Le paramètre effectif est du type PARTABELEMENT, tableau parallèle dont les éléments sont du type TABELEMENT. Il y a autant d'activations parallèles que d'éléments dans le type indice PROCESSOR.

Bibliographie

a)Tris et divers parallèles

Ban81 J.P. Banâtre, M. Banâtre
Parallel structures for vector processing *Conpar 81, Conference. on analysing problem classes and programming for parallel computing, Juin 81, p.p. 101-104*

Bat68 K.E. Batcher
Sorting networks and their applications *Spring Joint Computer Conference, 1968, p.p. 307-314*

Bau78 G. Baudet, D. Stevenson
Optimal sorting algorithms for parallel computers *I.E.E.E. Transactions on Computers, vol. C-27, N°1, Janvier 1978*

Ben65 K.E. Benes
Mathematical theory of connecting networks and telephone traffic *Academic Press, New York, 1965*

Flo64 R.W. Floyd
Algorithm 245 *C.A.C.M., Vol. 7, p. 701, Decembre 1964*

Fly72 M.J. Flynn
Some computer organization and their effectiveness *I.E.E.E. Transactions on Computers, Vol. C-21, N°9, Septembre 1972, p.p. 948-960*

Gav75 F. Gavril
Merging with parallel processors *C.A.C.M., Octobre 1975, Vol.18, N° 10, p.p. 588-591*

Hoa62 C.A.R. Hoare
Quicksort *Computer Journal, Vol. 5, N°1, p.p. 10-15, 1962*

Hir78 D.S. Hirschberg
Fast parallel sorting algorithms *C.A.C.M., Août 1978, Vol. 21, N°8, p.p. 657-661*

Knu73 D.E. Knuth
The art of computer programming, Vol. 3: Sorting and searching *Reading, M.A., Addison-wesley Publishing company*

Kun77 H.T. Kung, D. Stevenson
A software technique for reducing the routing time on a parallel architecture with a fixed interconnexion network *High speed computer and algorithm organization, p.p. 423-433, Academic Press, N.Y., 1977*

Led81 H.F. Ledgard
Ada, an introduction & Ada reference manual *Springer Verlag ed., New york, 1981*

Len78 J. Lenfant
Parallel permutations of data : A Benes network control algorithm for frequently used permutations *I.E.E.E. Transactions on Computers, Vol. C-27, p.p. 637-647, Juillet 1978*

Mar71 W.A. Martin
Sorting *Computing Surveys, Vol. 3, N°4, Decembre 1971, p.p. 147-174*

Mul75 D.E. Muller, F. Preparata
Bounds to complexity of networks for sorting and switching *Journal of the Association for Computing Machinery, Vol. 22, N°2, Avril 1975, p.p. 195-201*

Nas79 D. Nassimi, S. Sahni
Bitonic sort on a mesh-connected parallel computer *I.E.E.E. Transactions on Computers, Vol. C-27, N°1, Janvier 1979*

Orc74 S.E. Orcutt
Computer organization and algorithms for very high speed computations *PH. D. Dissertation, Comput. Sc. Dpt., Stanford Univ., Septembre 1974, Chap. 2, p.p. 20-23*

Orc76 S.E. Orcutt
Implementation of permutation functions in Illiac IV type of computers *I.E.E.E. Transactions on Computers, Vol. C-25, N°9, p.p. 929-936, Septembre 1976*

Pre78 F.P. Preparata
New parallel sorting schemes *I.E.E.E. Transactions on Computers, Vol. C-27, N°7, Juillet 1978, p.p. 669-673*

Sto71 H.S. Stone
Parallel processing with the perfect shuffle *I.E.E.E. Transactions on Computers, Vol. C-20, N°2, Fevrier 1971, p.p. 153-161*

Sto78 H. S. Stone
Sorting on STAR *I.E.E.E. Transactions on Soft. Eng., Vol. SE-4, N°2, Mars 1978, p.p. 138-146*

Tho76 C.D. Thompson, H.T.Kung
Sorting on a mesh-connected parallel computer *Carnegie-Mellon Univ. Report, Mars 1976 & C.A.C.M., Vol. 20, Avril 1977, p.p. 263-271*

Val75 L.G. Valiant
Parallelism in comparison problems *SIAM J. Comp., Vol. 4, N°3, Septembre 1975, p.p. 348-355*

b) Architectures

BSP Burroughs corporation
Burroughs Scientific Processor *Burroughs Corp. Publication N°4001564*

CDC STAR 100 R.G. Hintz, D.P. Tate
Control Data Star 100 Processor Design *COMPCON 72 Digest p.p. 5-8*

CRAY-1	R.M. Russel **The CRAY-1 computer system** *C.A.C.M., Vol. 21, N°1, Janvier 1978, p.p. 63-72*
EGPA	W. Händler, R. Klar, H. J. Schneider **A general purpose array with a broad spectrum of applications** *1st Workshop on Computer Architecture, Erlangen, Mai 1975, Springer Verlag, Berlin, 1976, p.p. 311-334*
ICL DAP	D. Parkinson **Technical description of the distributed array processor** *Document N° AP2 d'ICL, Novembre 1976*
Illiac IV	G.H. Barnes & al. **The Illiac IV computer** *I.E.E.E. Transactions on Computers, Vol. C-27, p.p. 746-757, Août 1968*
MPP	K.E. Batcher **Architecture of a massively parallel processor** *The 7th Annual Symposium On Computer Architecture, La Baule, Mai 80, p.p. 168-173*
NSS	Burroughs corporation **Navier-Stokes Solver -Numerical aerodynamic simulation facility -Preliminary study** *Ames Research Center -Burroughs Corp. Pub., Octobre 1977*
Phœnix	G. Feierbach, D.K. Stevenson **The Phœnix array processing system** *Institute For Advanced Computation, Sunnyvale, Californie, Novembre 1978*
Propal II	A. Gandossi **Le Calculateur Parallèle et Associatif Propal II** *Proceedings of the 1st European Conference on Parallel and Distributed Processing, Toulouse, Février 1979, p.p. 57-76*
TI ASC	W.J. Watson. **The TI ASC : A highly modular and flexible super computer architecture** *Proceedings FJCC, Vol. 41, 1972*

PI 147 **Deux files d'attente à capacité limitée en tandem**
J. Pellaumail, J. Boyer , 19 pages ; *Juillet 1981*

PI 148 **Programme de classification hiérarchique : 1) Méthode de la vraisemblance des liens, 2) Méthode de la variance expliquée**
I.C. Lerman , 113 pages ; *Juin 1981*

PI 149 **Convergence des méthodes de commande adaptative en présence de perturbations aléatoires**
J.J. Fuchs , 46 pages ; *Juillet 1981*

PI 150 **Construction automatique et évaluation d'un graphe d'«implication» issu de données binaires, dans le cadre de la didactique des mathématiques**
H. Rostam , 112 pages ; *Juin 1981*

PI 151 **Réalisation d'un outil d'évaluation de mécanismes de détection de pannes]-]Projet Pilote SURF**
B. Decouty, G. Michel, C. Wagner, Y. Crouzet , 59 pages ; *Juillet 1981*

PI 152 **Règle maximale**
J. Pellaumail , 18 pages ; *Septembre 1981*

PI 153 **Corrélation partielle dans le cas « qualitatif »**
I.C. Lerman , 125 pages ; *Octobre 1981*

PI 154 **Stability analysis of adaptively controlled not-necessarily minimum phase systems with disturbances**
Cl. Samson , 40 pages ; *Octobre 1981*

PI 155 **Analyses d'opinions d'instituteurs à l'égard de l'appropriation des nombres naturels par les élèves de cycle préparatoire**
R. Gras , 37 pages ; *Octobre 1981*

PI 156 **Récursion induction principle revisited**
G. Boudol, L. Kott , 49 pages ; *Décembre 1981*

PI 157 **Loi d'une variable aléatoire à valeur R^+** réalisant le minimum des moments d'ordre supérieur à deux lorsque les deux premiers sont fixés
M.Kowalowka, R. Marie , 8 pages ; *Décembre 1981*

PI 158 **Réalisations stochastiques de signaux non stationnaires, et identification sur un seul échantillon**
A. Benveniste J.J. Fuchs , 33 pages ; *Mars 1982*

PI 159 **Méthode d'interprétation d'une classification hiérarchique d'attributs-modalites pour l'«explication» d'une variable ; application à la recherche de seuil critique de la tension artérielle systolique et des indicateurs de risque cardiovasculaire**
B. Tallur , 34 pages ; *Janvier 1982*

PI 160 **Probabilite stationnaire d'un réseau de files d'attente multiclasse à serveur central et à routages dépendant de l'état**
L.M. Le Ny , 18 pages ; *Janvier 1982*

PI 161 **Détection séqentielle de changements brusques des caractéristiques spectrales d'un signal numérique**
M. Basseville, A. Benveniste , pages ; *Mars 1982*

PI 162 **Actes regroupés des journées de Classification de Toulouse (Mai 1980), et de Nancy (Juin 1981)**
I.C. Lerman , 304 pages ;

PI 163 **Modélisation et Identification des caractéristiques d'une structure vibratoire : un problème de réalisation stochastique d'un grand système non stationnaire**
M. Prévosto, A. Benveniste, B. Barnouin , 46 pages ; *Mars 182*

PI 164 **An enlarged definition and complete axiomatization of observational congruence of finite processes**
Ph. Darondeau , 45 pages ; *Avril 1982*

PI 165 **Accès vidéotex à une banque de données médicales**
A. Chauffaut, M. Dragone, R. Rivoire, J.M. Roger , 25 pages ; *Mai 1982*

PI 166 **Comparaison de groupes de variables définies sur le même ensemble d'individus**
B. Escofier, J. Pages , 115 pages ; *Mai 1982*

PI 167 **Transport en circuits virtuels internes sur réseau local et connexion Transpac**
M. Tournois, R. Trépos , 90 pages ; *Mai 1982*

PI 168 **Impact de l'intégration sur le traitement automatique de la parole**
P. Quinton , 14 pages ; *Mai 1982*

PI 169 **A systolic algorithm for connected word recognition**
J.P. Banâtre, P. Frison, P. Quinton , 13 pages ; *Mai 1982*

PI 170 **A network for the detection of words in continuous speech**
J.P. Banâtre, P. Frison, P. Quinton , 24 pages ; *Mai 1982*

PI 171 **Le langage ADA : Etude bibliographique**
J. André, Y. Jégou, M. Raynal , 12 pages ; *Juin 1982*

PI 172 **Comparaison de plusieurs variables : 2ème partie : un exemple d'application**
B. Escofier, J. Pajès , 37 pages ; *Juillet 1982*

PI 173 **Unfold-fold program transformations**

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique